**Design Patterns: Abstract Factory Pattern in TypeScript**

Master the Abstract Factory Pattern and Easily Handle the Creation of Objects.

Welcome to the **Design Patterns in TypeScript** series, which introduces some useful design patterns in web development using TypeScript.

Previous articles are as follows:

Strategy Pattern in TypeScript

Chain of Responsibility Pattern in TypeScript

Observer Pattern in TypeScript

Template Method Pattern in TypeScript

Adapter Pattern in TypeScript

Factory Method Pattern in TypeScript

Abstract Factory Pattern in TypeScript

Design patterns are very important for web developers and we can write better code by mastering them. In this article, I will use **TypeScript** to introduce the **Abstract Factory Pattern.**

The abstract factory pattern provides an interface for creating a series of related or interdependent objects without specifying their concrete classes.

In the [**factory method pattern**](https://javascript.plainenglish.io/design-patterns-factory-method-pattern-in-typescript-c4c3047a6289), a specific factory is responsible for producing specific products, each specific factory corresponds to a specific product, and the factory method is also unique. In general, there is only one factory method or a group of overloaded factory methods in a specific factory. But sometimes we need a factory that can supply multiple products, not a single product.

In the above picture, I simulated the user’s car purchase process. Bytefer ordered the SuperX01 from the SuperX factory, and the factory produced it according to the model corresponding to the SuperX01 and delivered it to bytefer after the production was completed. Chris1993 ordered SuperX02 from the same SuperX factory, and the factory produced it according to the model corresponding to SuperX02 and delivered it to chris1993 after production was completed.

Let’s look at how to use the **abstract factory** to describe the process of producing a given model of car in a vehicle factory.

In order to better understand the following code, let’s first look at the corresponding UML diagram:

The abstract factory pattern includes the following roles:

* **Product(Vehicle)**: abstract product
* **Concrete Product(SuperX01)**: concrete product
* **Factory(SuperXFactory)**: abstract factory
* **ConcreteFactory(ConcreteSuperXFactory):** concrete factory

Next, we define an abstract class Vehicle and its two subclasses SuperX01 and SuperX02 to represent different types of vehicles.

Then, we define the SuperXFactory class to represent the vehicle factory. This abstract factory contains abstract methods for producing SuperX01 and SuperX02 model vehicles.

Based on the SuperXFactory abstract class, we define the ConcreteSuperXFactory factory classe for the production of **SuperX01** and **SuperX02** models of vehicles:

After creating the ConcreteSuperXFactory factory classe, we can start producing vehicles:

When you successfully run the above code, the terminal will output the following result:

We have introduced the [**factory method pattern**](https://javascript.plainenglish.io/design-patterns-factory-method-pattern-in-typescript-c4c3047a6289) in the previous article, so what is the difference between it and the abstract factory pattern?

The biggest difference between the abstract factory pattern and the [**factory method pattern**](https://javascript.plainenglish.io/design-patterns-factory-method-pattern-in-typescript-c4c3047a6289)is that **the factory method pattern is aimed at a product hierarchy, while the abstract factory pattern needs to face multiple product hierarchy structures, and a factory hierarchy structure can be responsible for multiple different product hierarchy structures.**

The creation of the product object. When a factory hierarchy can create all objects in a product family belonging to different product hierarchy, the abstract factory pattern is simpler and more efficient than the factory method pattern.